

AFRL-IF-RS-TR-2002-281
Final Technical Report
October 2002



SCALABLE INTRUSION DETECTION AND RESPONSE FRAMEWORK

Odyssey Research Associates, Inc.

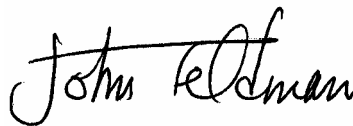
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-281 has been reviewed and is approved for publication.

APPROVED:

A handwritten signature in black ink, appearing to read "John Feldman". The signature is fluid and cursive, with a large initial "J" and a long, sweeping underline.

JOHN FELDMAN
Project Engineer

FOR THE DIRECTOR:

A handwritten signature in black ink, appearing to read "Warren H. Debany, Jr.". The signature is fluid and cursive, with a large initial "W" and a long, sweeping underline.

WARREN H. DEBANY, Jr., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE			<i>Form Approved</i> <i>OMB No. 074-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE Oct 02	3. REPORT TYPE AND DATES COVERED Final Apr 97 – Apr 99	
4. TITLE AND SUBTITLE SCALABLE INTRUSION DETECTION AND RESPONSE FRAMEWORK			5. FUNDING NUMBERS C - F30602-97-C-0140 PE - 61102F PR - 2301 TA - 01 WU - 04	
6. AUTHOR(S) Mark E. Reilly				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Odyssey Research Associates, Inc. Cornell Business & Technology Park 33 Thornwood Drive, Suite 500 Ithaca, NY 14850			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/IFGB 525 Brooks Rd Rome, NY 13441-4505			10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2002-281	
11. SUPPLEMENTARY NOTES AFRL Project Engineer: John Feldman, IFGB, 315-330-2664				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.				12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 Words) This effort developed a prototype scalable intrusion detection and response framework that hosts a set of intrusion detection and response technologies that demonstrate scalability in a high-assurance environment and ease of deployment as well as overcoming some of the limitations of traditional intrusion detection systems. This environment allows a developer to build an intrusion detection system without having to be concerned about the low-level, system-dependent details such as how to access built-in operating systems and hardware security functions, how to make a process on one computer communicate to a process on another computer, how to deploy an intrusion detection system, etc. This SIDF framework provides an open environment that sustains a wide variety of intrusion detection agents. The open architecture of this framework allows for a varying set of agents to be developed by a wide range of organizations.				
14. SUBJECT TERMS Intrusion detection, Software agent framework				15. NUMBER OF PAGES 43
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Abstract

Traditionally, large monolithic intrusion detection systems have been developed that operate on a host computer. These monolithic systems perform the task of monitoring, data gathering, data manipulation, data storage, decision making, and alarming. However, several significant problems result from using this approach to intrusion detection. These problems include:

- Large processing overhead is generated from monitoring activities, generation of audit records, the analysis and reduction of audit logs, generation of aggregate statistics, and communication and correlation of activities between networked systems.
- Storage space usage is often large and results in at least temporary reduction in available disk space.
- Typically, intrusion detection systems must be completely rebuilt from scratch to handle new scenarios.
- These monolithic approaches result in systems with a single point of failure and attack.
- Finally, scalability into large geographically diverse systems is limited.

We have developed a prototype scalable intrusion detection and response framework that hosts a set of intrusion detection and response technologies that demonstrate scalability in a high-assurance environment and ease of deployment as well as overcoming some of the limitations of traditional intrusion detection systems

Table of Contents

1.0 Summary.....	1
2.0 Introduction.....	2
2.1 Framework Introduction	2
2.2 Framework Operation	4
2.3 Framework Components	5
2.4 Framework Operation	7
2.5 SIDS Security.....	8
3.0 Methods, Assumptions, and Procedures.....	11
3.1 Technical Approach	12
3.2 SIDS Refinement	16
4.0 Results and Discussion.....	17
4.1 Deliverables	17
4.1.1 Framework Documentation	18
4.1.2 Deployment Center.....	18
4.1.3 Agent Guard	19
4.1.4 Directory Agent	20
4.1.5 Decision-Response Agent.....	20
4.1.6 Analysis Agents	21
4.1.6.1 NT-based Analysis Agent.....	21
4.1.6.2 Linux-based Analysis Agent.....	21
4.1.7 Reconnaissance Agents	22
4.1.7.1 TCP/IP Display Reconnaissance Agent.....	22
4.1.7.2 Traffic Analysis Reconnaissance Agent	23
4.1.7.3 Immunology-based Reconnaissance Agent	24
4.1.7.4 Linux Reconnaissance Agent.....	24
4.1.8 Storage Agent	25
4.2 The Future of SIDS	25
4.2.1 Provide the Agent as an ActiveX Control	25
4.2.2 Fault Tolerance Communication	25
4.2.3 Improve Security of Communication	26
4.2.4 Add More Specialized Reporting	26
4.2.5 Optimize the Code and the Libraries	27
4.2.6 Software-based PKCS-11 Implementation.....	27
4.2.7 Add Agent Types.....	27
4.2.8 Update Ethernet Capture Routines	27
4.2.9 Upgrade to Delphi 4.....	28
4.2.10 Use fewer third party components.....	28
4.2.11 Create a Custom Palette.....	28
4.2.12 Compress Files and Messages	28
4.2.13 Automatic Deployment.....	28
4.2.14 Automatic SIDS State Archival.....	29
4.2.15 Automatically Find Associated Agent.....	29
4.2.16 Upgrade Directory Agent	29
4.2.17 Replay Attack Protection.....	29
4.2.18 SIDS Incorporation	30

4.2.19	Send Initial Configuration Information	30
4.2.20	Deploy Additional Files with Agent.....	30
4.2.21	Add Auditing Functions	31
4.2.22	Framework State Tracking	31
4.2.23	Framework Traceability.....	31
4.2.24	Add Database Capabilities.....	31
4.2.25	Agent Information Archival	31
4.2.26	Reduce Agent Guard Visual Component.....	31
4.2.27	Agent Visual Component Modification.....	32
4.2.28	Reduce Agent Dependency.....	32
4.2.29	Add Deployment Management Console.....	32
5.0	Conclusions.....	33
5.1	Program Technologies	33
5.2	Future Programs	33
6.0	References.....	34

Figures

Figure 1. SIDF Agent Hierarchy.....	2
Figure 2. Physical Instantiation of SIDF.....	3
Figure 3. Framework Usage.....	5
Figure 4. Example Framework Environment.....	6
Figure 5. Deployment Center and Agent Guards.....	7
Figure 6. SIDF Hardware Encryption Devices.....	9
Figure 7. SIDF Security Implementation.....	10
Figure 8. Deployment Center Main Screen.....	19
Figure 9. Agent Guard Main Screen.....	19
Figure 10. Directory Agent.....	20
Figure 11. Decision-Response Agent.....	20
Figure 12. Linux Analysis Agent.....	21
Figure 13. TCP/IP Display Reconnaissance Agent.....	23
Figure 14. Traffic Analysis Reconnaissance Agent.....	24
Figure 15. Immunology Reconnaissance Agent.....	24

Tables

Table 1. SIDF Documentation Set.....	18
--------------------------------------	----

1.0 Summary

We have developed a Scalable Intrusion Detection and Response Framework. This framework was originally named SIDS. The name was later changed and presented externally as the Open Infrastructure for Scalable Intrusion Detection and Response due to the possibility of confusion with the similarly sounding CIDS (the Common Intrusion Detection Framework) that DARPA is developing. We have retained the SIDS acronym on all reports and documentation internally to maintain consistency throughout the effort. This report uses the term SIDS to describe the framework.

A dictionary defines the word “framework” as “a basic conceptual structure.” What we proposed was a structure that included example code, libraries, processes, and documentation for a development environment. This environment will allow a developer to build his/her own intrusion detection system without having to know about the low-level, system-dependent details such as how to access built-in operating system and hardware security functions, how to make a process on one computer communicate to a process on another computer, how to deploy an intrusion detection system, etc.

This framework provides an open environment that sustains a wide variety of intrusion detection agents. The open architecture of this framework allows for a varying set of agents to be developed by a wide range of organizations.

The hierarchically directed agents can be deployed and retracted throughout the network based on the perceived threat, operational conditions, mission, or other policy-based criteria. The framework allows for the development of new agents, the removal of old or obsolete agents, the modification of parameters of operating agents, as well as the dynamic deployment and retraction of agents.

This new framework provides a much-needed environment to encourage the development and deployment of a vast array of new intrusion detection and response technologies. We expect that the use of this framework will lead to the development of a wide range of intrusion detection and response solutions that interoperate. Without this framework, the intrusion detection field will continue to be inundated with monolithic, homogeneous, closed-architecture solutions that provide limited scalability.

2.0 Introduction

2.1 Framework Introduction

The Scalable Intrusion Detection and Response Framework (depicted as a hierarchy in Figure 1) can logically be represented by four distinct logical layers: the Decision-Response Layer, the Analysis Layer, the Reconnaissance Layer, and the Support Layer. Each layer plays a role in detection, alarm, response, deployment, scalability, and support for the framework.

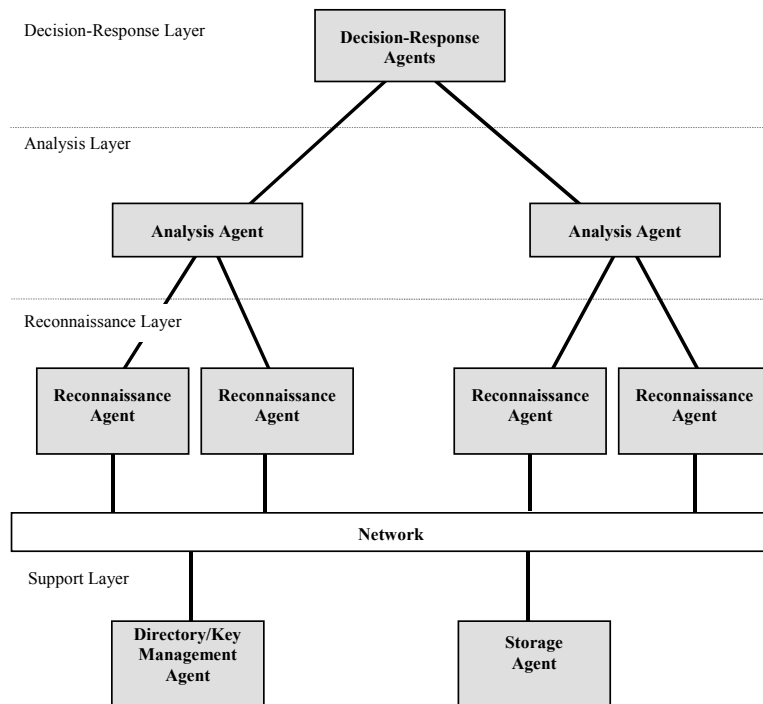


Figure 1. SIDF Agent Hierarchy.

The purpose of the four layers is to divide the typical intrusion detection and response paradigm into a set of more manageable components that can easily be distributed throughout a geographically diverse heterogeneous environment. Within each layer, agents are defined and each performs specific subfunctions for the entire intrusion detection system.

Each layer defines one or more types of agents. Several types of agents have been defined; the types currently available are Decision-Response, Analysis, Reconnaissance, Directory, and Storage. The type of agent defines its function within the framework. For example, a Reconnaissance type of agent is responsible for monitoring and reporting specific information to a superior agent. Whereas a Directory type agent maintains a directory that lists all the currently deployed agents within the framework. The agents are named based on their type. A Reconnaissance type of agent is known as simply a Reconnaissance Agent and a Directory type of agent is known as a Directory Agent, and so. The list of agents types included in the SIDF are

the Directory Agent, Decision-Response Agent, Analysis Agent, Reconnaissance Agent, and Storage Agent.

Note that the above figure only represents a conceptual view of the framework and not a physical view. All agents within the framework have network access and each agent may interact with other agents by sending messages to each other. The framework defines the hierarchy of communication rules. The following list represents some of the SIDF communication design restrictions:

- Reconnaissance Agents communicate to Analysis Agents
- Analysis Agents communicate with Decision-Response Agents
- All agents send and receive messages to and from the Directory Agent
- Any agent can send messages to the Storage Agent
- Etc.

Several message formats have been defined within the framework. These messages are used to do everything from reporting status, setting reporting hierarchy, changing configuration parameters, etc.

Consider an example physical instantiation of the framework. This example depicts a four-host framework configuration with a variety of deployed agents.

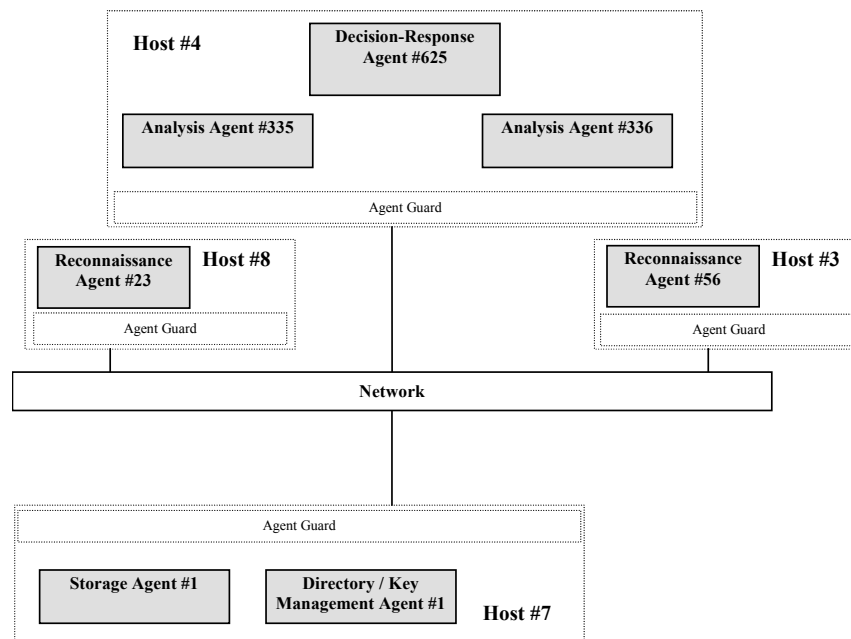


Figure 2. Physical Instantiation of SIDF.

2.2 Framework Operation

The SIDF framework developed on this project consists of a set of components known as “agents” and framework control applications. The framework control applications are the Deployment Center and the Agent Guard. These applications are not agents (as we defined the term for this project). The Deployment Center and Agent Guard handle agent deployment, messaging and data encryption. The Deployment Center acts as the control module for the framework, allowing instantiation of the framework, configuration of host encryption keys, and storage of agent executables. After configuration of encryption tokens, installation of the Deployment Center and Agent Guards, creation of agent executables, and general framework setup, the framework can be put into operation.

Before utilization of the SIDF, several steps must be taken to configure the deployment environment. The Deployment Center must be installed and a Deployment Center token must be created. Tokens must also be created for each host participating in the SIDF, and a set of agent executables needs to be built. Figure 3 illustrates how an agent is deployed in the SIDF (also known as the deployment process.)

Deployment Process:

- An agent is selected from the Agent Executables Database.
- A host is selected from the Host Database (a list of hosts assigned encryption keys).
- The selected agent executable is encrypted with the public key of the receiving host, in the figure this would be Host A.
- The encrypted agent executable (illustrated as a screenshot with a padlock on it) is transmitted to Host A from the Deployment Center.
- The Agent Guard on Host A receives the file, decrypts it, verifies the sender, and starts it executing on Host A.
- This process can be repeated several times, transmitting agents to the same or other hosts.
- Once the framework is populated with multiple agents, the hosts and agents will intercommunicate with framework-defined messages.

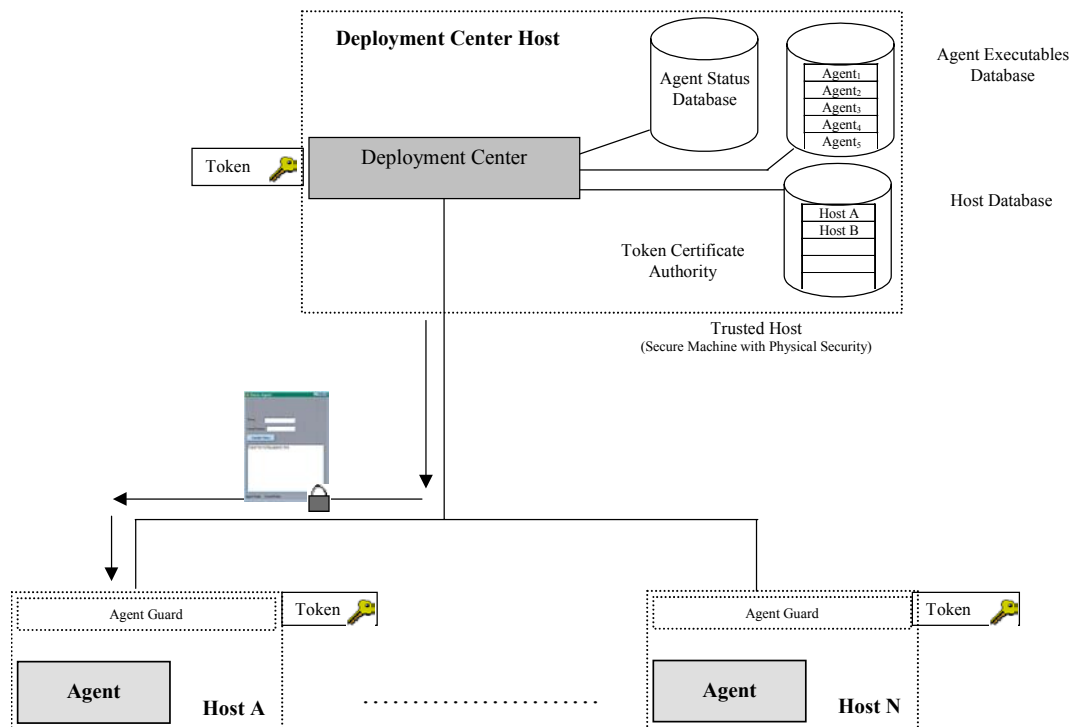


Figure 3. Framework Usage.

2.3 Framework Components

The SIDF consists of the following applications: the Deployment Center, Agent Guard, Directory Agent, Decision-Response Agent, Analysis Agent, Reconnaissance Agent, and Storage Agent. Some of these applications have been specialized and multiple versions delivered. For example, this project's requirements stated that four Reconnaissance Agents, and two Analysis Agents must be developed. Each agent has been customized with particular specialized functionality. Figure 4 illustrates several of the developed SIDF applications.

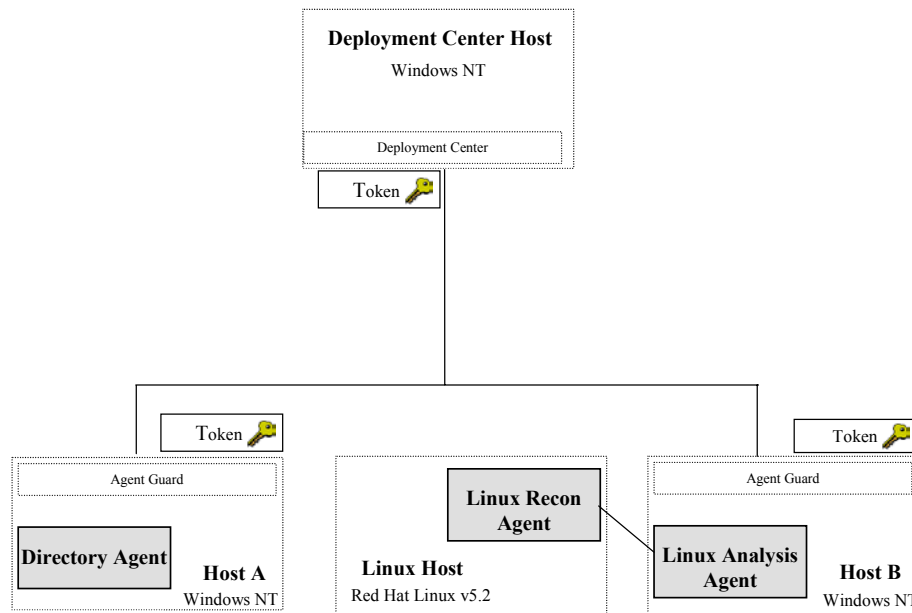


Figure 4. Example Framework Environment.

Several SIDF applications have been developed, some of which are illustrated in the figure above. A complete list follows, along with a brief description.

- **Deployment Center** – initiates the framework; configures, issues and maintains the encryption keys for the other framework hosts; and controls the deployment of agents to other hosts.
- **NT Agent Guard** – receives deployed agent executables, initiates the execution of the agent on the host, and controls all local and external framework messaging.
- **Base Agent** – represents the basic SIDF agent configuration and attributes that all other agents inherit.
- **Directory Agent** – maintains a list of agents deployed in the framework. Additionally this agent stores the public keys of the other hosts in the framework.
- **Decision-Response Agent** – receives reports from one or more Analysis Agents and can be configured to perform user-initiated responses based on information received.
- **NT Analysis Agent** – receives reports from one or more Reconnaissance Agents. The Analysis Agent need to “understand” the information that a Reconnaissance Agent sends. Thus, the pairing of Analysis Agents to Reconnaissance Agents should be based on similar monitoring activities (e.g., TCP/IP Reconnaissance Agents should be paired with TCP/IP Analysis Agents). The Analysis Agents also report information periodically to a Decision-Response Agent.
- **Linux Analysis Agent** – configures, controls, and receives reports from the Linux Reconnaissance Agent. The Analysis Agent also reports information periodically to a Decision-Response Agent.
- **Ethernet Capture (TCPSniffer) Control** – provides a programming interface to an installed TCP/IP protocol capture stack that allows the TCP/IP packets’ header and content to be examined.

- TCP/IP Display Reconnaissance Agent – displays information associated with the Ethernet traffic passing through the network segment to which the host system is connected. This agent uses the TCPSniffer stack to capture TCP/IP packets.
- Immunology-based Reconnaissance Agent – monitors incoming packets and compares the packet content to a self-database. Packets that do not meet the database's criteria are flagged and a message sent to an Analysis Agent. This agent uses the TCPSniffer stack to capture TCP/IP packets.
- Traffic Analysis Reconnaissance Agent – analyzes and reports to an Analysis Agent such network traffic data as the number, type, and content of packets that it has monitored during a specified time period. This agent uses the TCPSniffer stack to capture TCP/IP packets.
- Linux Reconnaissance Agent – runs under Red Hat Linux v5.2. This agent communicates directly to the Linux Analysis Agent running on an NT computer. This Analysis Agent is similar to a proxy server--it has been specifically created to communicate with the Linux Reconnaissance Agent. This agent monitors changes to system-specific files and reports status information to the Analysis Agent.
- Storage Agent – acts as an event-logging agent for the framework. Any SIDF component can issue a message to be logged by a Storage Agent.

2.4 Framework Operation

Each of the agent components provides the SIDF framework's intrusion detection and response functions. The Deployment Center and the Agent Guards (Figure 5) make up the control and security components. The Deployment Center and Agent Guard control the starting up of every agent in the SIDF and provide the communications between components.

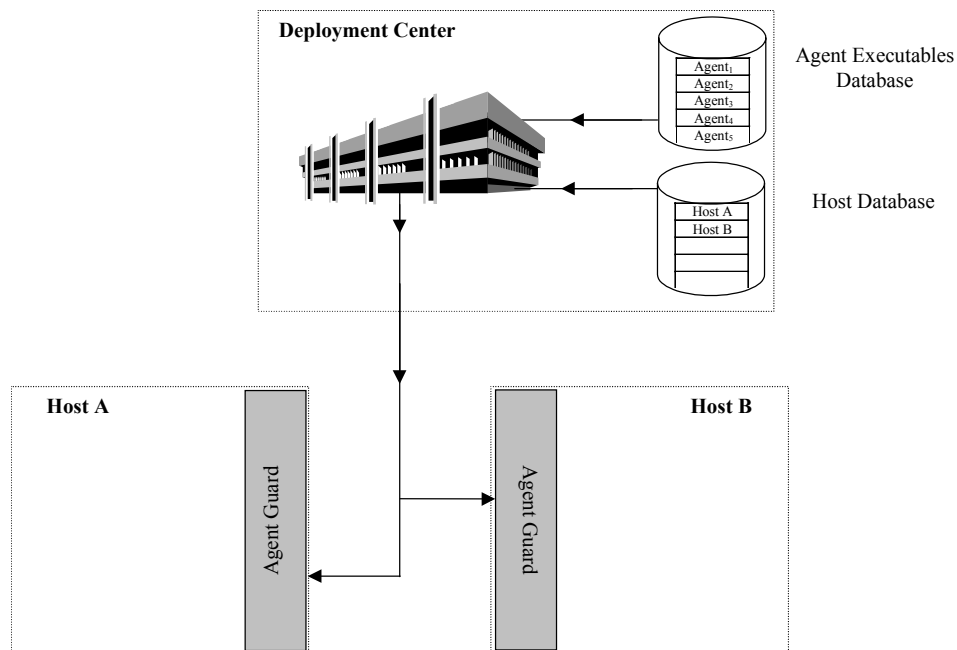


Figure 5. Deployment Center and Agent Guards.

The Deployment Center transmits (deploys) agent binaries to hosts for execution. The Deployment Center creates and maintains the security keys for all hosts. It can issue a limited set of remote commands to SIDF components. The only entity on the network and within the SIDF that the Deployment Center directly communicates with is the remote host's Agent Guard. At the SIDF's startup time, the Deployment Center knows of all hosts that can possibly participate in the framework.

The Agent Guard is a process that runs on each SIDF host computer, listening for the SIDF Deployment Center or other Agent Guards to issue SIDF commands to it on a predefined port before responding appropriately. Any host participating in the SIDF must have an Agent Guard running on it. However, there is only one Deployment Center in any instance of the SIDF.

Figure 5 illustrates a simple SIDF configuration consisting of three computers. The Deployment Center runs on a trusted host. It has a database of host information and a database of agent executables that have been previously created. These agents include Decision-Response Agents, Analysis Agents, Reconnaissance Agents, Directory/Key Management Agents, and Storage Agents. Several of the agents are actually specialized applications--that is, agent code is already written and requires little to no configuration (namely the Decision-Response Agents, Storage Agents, and Directory/Key Management Agents). Others are customized agents (specialized Analysis Agents and Reconnaissance Agents) that developers have created to monitor and analyze particular system or network parameters. Many different specialized agents coexist in any SIDF instantiation; the number, type, and function of these agents depends on the type of information monitored and the rules programmed in the agent's code.

The Deployment Center allows the SIDF administrator to assign any agent from the agent database to execute on a targeted host. The targeted host must run an Agent Guard to communicate with the Deployment Center. In this example, either Host A or Host B could be selected as a target host. An agent binary selected from the Deployment Center's agent database would be transferred to the Agent Guard on the host; the agent program would then start executing.

The Agent Guards are responsible for instituting the security features that SIDF agents require. We have designed the SIDF in this fashion so that the complexities of host-based security could be centrally controlled, thus making the SIDF more secure. To this end, the Agent Guard acts as a gateway/intermediary between agents on different hosts.

2.5 SIDF Security

In our Scalable Intrusion Detection and Response Framework, we do not assume any inherent security in the network to which the SIDF is applied. In particular, we admit the possibility of malicious entities manipulating either a) agent-to-agent communications or b) the agents themselves. Given this, the SIDF must inherently provide security for intrusion detection agent deployment and operations. Therefore, we designed security into our framework from the beginning, as opposed adding it as an afterthought.

The SIDF provides protection of functions and communication using hardware-based security, because the level of protection these devices provided was deemed essential to the framework

based on the types of information that it could potentially access. Two types of hardware encryption devices are employed in the SIDF. These hardware devices are the Chrysalis-ITS Luna 2 token and Datakey's SignaSURE Datakey (both pictured below).



Figure 6. SIDF Hardware Encryption Devices.

Originally, the SIDF was designed and built exclusively using Luna2 tokens. There is a cost associated with adding hardware-based encryption devices to any system, but Luna2 tokens are especially expensive. A single Luna2 PCMCIA security token costs approximately \$800 and requires the installation of a PCMCIA reader (because one usually isn't included in most desktop PC configurations).

At the start of this project, the Luna 2 tokens cost about \$300, with an expected cost that would decrease to less than \$100 per token as the technology matured. Since the manufacturer was unsuccessful in selling its product in the client market, the company increased the price and targeted the higher-end server market.

To address this issue, we have investigated alternative hardware solutions. We found our solution with Datakey. This hardware device is a combination reader and key (or card) that plugs into a standard serial port of a PC--there is no additional hardware required. The performance (speed of encryption/decryption operations) is less than a Luna 2 token, but is acceptable. The Datakey devices provide similar PKCS functionality and cost about \$100 each.

We incorporated Datakey into the framework by providing the option of having an NT Agent Guard use a Luna 2 token or a Datakey as the host's encryption device. The Deployment Center requires one Luna 2 token, but can issue Luna or Datakey NT Agent Guard keys.

Figure 7 illustrates the security implementation used in the SIDF.

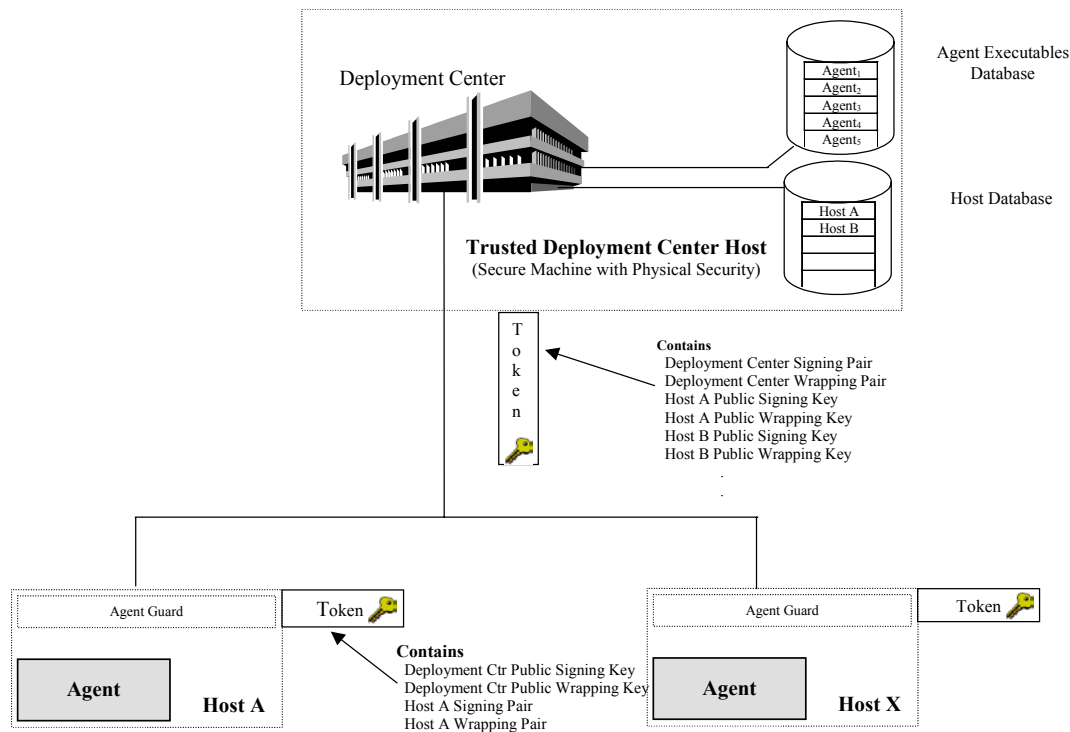


Figure 7. SIDS Security Implementation.

As shown in Figure 7, the Deployment Center host has a cryptographic token (Luna 2) installed in it. The host is a physically secured, trusted computer. Before SIDS instantiation, tokens (Luna or Datakey) are created at the Deployment Center for each host on the network. Each host will have one token containing a signing and a wrapping pair of keys. A copy of the public signing and wrapping key will be maintained in the token on the Deployment Center host.

When an agent is first registered with the Directory Agent, copies of the local host's public keys are sent to the Directory Agent. The Directory Agent maintains a table of public keys that the SIDS uses. Messages have been defined in the SIDS message grammar for retrieval and update of the Directory Agent's public key list. In addition, public keys can be transmitted from the Deployment Center to a specified host. The transmittal of public keys within the SIDS is important because all messages between hosts are encrypted specifically from the sending host to the receiving host. Hosts cannot communicate with each other until they have exchanged public keys. Once keys are exchanged during a SIDS session, the keys are stored permanently on the hardware encryption token and can be used in the future. Note that private keys are never removed from the hardware token, nor are they ever exposed. This is one of the hardware device's main strengths, along with the necessity that users of the framework need a password to enable the SIDS's functions.

3.0 Methods, Assumptions, and Procedures

This effort's main focus was the development of a mostly PC-based framework that could be useful in the field of Intrusion Detection. We have observed that at the time of our initial proposal for this effort, most intrusion detection systems (IDSs) were built on Unix systems and were generally large and required many system resources to operate.

Our goal was to produce a framework that would lead to the quicker development of new intrusion detection and response systems by first building a framework that could be used on cheaper and more easily configurable Windows-based PCs. Building a framework with reusable and interoperable components would allow new/future IDS builders the ability to concentrate their development on the problem at hand (building new algorithms to detect the new attack) instead of "reinventing the wheel" each time (determining how to capture the data, thinking about how to securely store the monitored data, determining a method to effectively store the data, etc.)

We had to make several assumptions in our initial prototype framework. First and foremost, we selected Windows NT 4.0 as the main targeted environment. Windows NT provides several built-in security features and has a quickly increasing user base. As mentioned above, several Unix versions of IDSs exist but not that many for PCs. We wanted to increase the number of ID systems that could be created for the PC. Also since several companies are rapidly purchasing PC systems and transferring functions from larger workstations/minicomputers to PCs, the risk to PC systems is greatly increasing as PCs become targets of more types of attacks.

Secondly, we wanted to assure any would-be user and IDS developer that the framework that we created would be secure. Therefore, we built hardware-based encryption into the system. The framework is controlled using messages to and from multiple systems. Each message and request is encrypted so that the message cannot be deciphered outside of the application. Moreover, each message is signed so that it can be authenticated when received.

Third, to create a "developer-friendly" environment, we selected a visual development tool to create the framework. The final framework consists of base forms and code that developers can modify as required to build new customized ID agents and systems. The environment that we have used is Borland Delphi Professional v3.02.

As with any development framework, designers face some inherent constraints. The following are the constraints that we identified within the SIDS:

- Requires an order of installation/startup of framework.
- Is developed in Borland Delphi Professional v3.02 and requires the developer to continue to develop in Delphi.
- Uses third-party components.

Based on the above constraints and the prototyped version of the SIDS, the following limitations are recognized:

- Limited portability.

- Limited number of currently developed agents.
- Sophistication of developed agents needs to be improved.
- Greater detail needs to be incorporated into reporting functions.
- Additional messages and configuration functions should be added.

3.1 Technical Approach

At the start of this project, we reviewed the technology available within Odyssey Research and the Intrusion Detection community. We analyzed the requirements that we needed to incorporate into the framework, and created an initial design of the SIDS intrusion detection architecture.

During this time, DARPA began discussions on the creation of a Common Intrusion Detection Framework (CIDF) standard. We assisted the committee by submitting a set of intrusion detection APIs to the CIDF working group for discussion and incorporation. We also had an ORA staff member sit on the committee during the development of the CIDF standard.

The CIDF standard's goal and its set of APIs is to allow open access to intrusion detection analysis modules, database records, input event monitors, and intrusion response modules. Since this goal was very similar to ours, our cooperation benefited both the CIDF and the SIDS by incorporating our SIDS design ideas into the CIDF standard. Thus, the SIDS would be compatible with the CIDF as the CIDF standard matures.

It is also important to differentiate SIDS from CIDF. When we originally proposed the SIDS, CIDF did not exist. The CIDF grew out of DARPA's increasing need to have multiple contractors interface to each other. ORA was in the initial meetings on CIDF, and we continue to play a lead role in the standard's development. The main distinction between the two is that CIDF is a standard and whereas SIDS is an implementation (of a standard.) Unfortunately, the CIDF was in continual development during the SIDS project's life and continues to change today. Keeping this in mind, the SIDS has been designed with a flexible message grammar that could be adapted. In addition, most of the SIDS components map directly to CIDF components.

One of the first things that we did in the implementation of the SIDS was to prototype agent-to-agent communication based on our draft protocol specification. We created a Base Agent class to support a minimum set of the properties, including basic identification information, client/server support, cryptographic functions, and directory access behaviors. We then drafted versions of Reconnaissance Agent, Analysis Agent, and Decision-Response Agent classes. All this initial work was done in C++.

A testbed of two computers was created in which we could deploy the agents and test the basic functions of each class.

After some initial prototype work, we started part of the large SIDS documentation effort. We wrote the draft specification for the protocol to be used for intra-agent communication. The protocol, our secure agent management protocol, was based on TCP/IP and allowed agents to communicate and interact. The agent communication requirements followed those specified in the SIDS documentation.

We defined the agent communication hierarchy such that all agents would be directly accessible over the network. However, framework communication would be restricted to specified channels, such as Decision-Response Agent to Analysis Agent, Analysis Agent to Decision-Response Agent, Reconnaissance Agent to Analysis Agent, etc.. Other channels, such as Reconnaissance Agent to Decision-Response Agent, were not allowed.

In our original proposal, we identified several types of agents. Two new agents were introduced: the Storage Agent and the Directory/Key Management Agent. We added these agents to provide a method to audit and keep track of each agent within the network.

The Scalable Intrusion Detection and Response Framework System Specification document was written and contained all the framework's requirements. We used this document as the basis of the framework and prototype development for the SIDF project. For ease of requirements tracking, all the SIDF requirements were specified in table format and uniquely numbered.

The Scalable Intrusion Detection and Response Framework Architecture Specification was written and specified the structure of the SIDF framework and the agents that it would contain. Originally, the framework only consisted of five agents:

- Decision-Response Agent
- Analysis Agent
- Reconnaissance Agent
- Storage Agent
- Directory/Key Management Agent

The interaction and functions of each agent were described within this specification. An object class structure was designed to illustrate the functions and properties of each type of agent.

The Agent Guide for Development provided information that would allow a software designer/programmer to use the framework to implement each of the agent types. The document contained the definitions of the agent classes along with the required C++ include files.

Agent specifications were designed using a C++ object-oriented paradigm. Each agent type was designed as a separate class. Creation of an agent was accomplished through instantiation of the specified agent class. Each agent class inherited its lowest-level data structures and functions from a base class. We later wrote the Agent Guide for Development document after we migrated the development environment from C++.

The next step was to concretely define the processes of agent communication, initialization, deployment, and retraction. We could then better prototype a method for these functions.

We found that several difficulties existed when setting up the SIDF environment. Particularly, configuring agents when each starts up may pose a problem since the SIDF hierarchy requires parent-to-sibling communication. The problem was resolved by imposing an 'ordering' to the process of starting up and shutting down agents. The ordering problem has been mostly

eliminated in the final SIDF version. The main framework startup requirement that currently exists is that the Directory Agent must be the first agent deployed in a SIDF instantiation and other agents must be notified of the Directory Agent's location.

The ease of use, and the ability to reuse agent code, were very important requirements in the design of the SIDF. The initial prototype efforts focused on representing each type of agent as an object class (i.e., Reconnaissance Agent class, Analysis Agent class, Decision-Response Agent class, etc.). We then implemented the abstract view of agent objects as C++ classes. Although this representation was very common in the software development world, it required a developer of new agents to know intimate knowledge of the C++ agent class structures and the use of the behaviors associated with them.

To make the technology developed from this project more useable, we started to look at the ability to turn each agent object into a *component* such that it could just be “dragged-and-dropped” onto a form in a visual environment such as Borland Delphi or Microsoft Visual Basic.

The SIDF System Design Description document described the SIDF design decisions, the SIDF architectural design, and the detailed design needed to implement the framework. The design was described by looking at, and defining each of the agents that was to be implemented. A set of agents was to be developed at each level of the hierarchy of the SIDF framework: one Decision-Response Agent, two Analysis Agents, and four Reconnaissance Agents. Additionally, we needed to design and develop at least one of each support agent (Storage Agent and Directory/Key Management Agent), too. Also we specified class hierarchies, and defined the specific properties, methods, and events available for each class of agent.

We found several inherent problems in the SIDF's proof of concept version's internal design with respect to the deployment and retraction processes. These problems related to the ability to deploy and start executing software on remote computers. Simply stated, it was difficult to figure out a way to pass a program to a remote computer and have it start executing. First, this process could not be done securely since the destination host did not have any encryption (keying) information. And secondly how would the destination host wait for, accept, and start running the transmitted agent software? Assuming that we could work out the initial deployment problems, how would the agent be configured and controlled?

Our first approach was a manually oriented deployment scheme and was plagued with potential faults. A new design was proposed that had many benefits, including improved security and simplification of agent development. At the very top level of the newly proposed SIDF (above the agent level), we introduced the SIDF Deployment Center and the Agent Guards. The Deployment Center would be responsible for transmitting commands and agent binaries to hosts for ultimate execution. At framework startup time, the Deployment Center would know of all hosts that could possibly participate in the framework. The Deployment Center would issue commands directly to an Agent Guard. There would only be one Deployment Center in any instance of the SIDF.

The Agent Guards would passively listen for commands from the Deployment Center. Any host that would participate in the SIDF would have an Agent Guard running on it. All SIDF

communication security would be handled in a centralized fashion at this level. The SIDF provided protection of agent functionality and communication using hardware-based security at the Deployment Center/Agent Guard level.

We employed a rapid prototyping approach to develop the new NT Agent Guard and Deployment Center. We first started building an NT Agent Guard prototype. The Agent Guard was developed in Borland Delphi and used SMARTCrypt™ for data and message encryption. Functions were created to construct, sign, and accept Deployment Center messages. The packaged messages were received, verified, unpackaged, and executed.

The deployment messages followed the design as defined in the SIDF Framework Detailed Design document. Example messages were constructed and the encryption functions were written to test basic functionality. Signing and wrapping functions used 1,024 or 2,048-bit RSA keys. DES was used for session keys. We then implemented several of the security functions and incorporated the basic deployment functions into the Deployment Center.

Next, we started preliminary coding of an NT-based Ethernet traffic-capturing component. We drafted the design of an NT-based Ethernet traffic-capturing (TCPSniffer) component, and we created a design document. Many of the demonstration Reconnaissance Agents rely on passive collection of network packets. This component provided the ability to selectively capture and parse the network packets that meet specified criteria. The selection criteria can be changed during runtime agent reconfiguration.

An initial Base Agent was then drafted in Delphi and a Directory Agent was created to inherit the Base Agent. We used the Directory Agent for testing and building-up the Base Agent. Agent-to-agent communication was drafted into the Base Agent, and we tested it with the example Directory Agent. From these prototypes, we created each of the other agents.

We then prototyped the complete Deployment Center. The design and implementation of a Unix version of a Reconnaissance Agent followed. We then tested the deployment function. An example agent executable was encrypted, transmitted from the Deployment Center to a second system, received, decrypted, verified, and executed on the second system. At this point, we knew that our deployment concept worked.

As our work continued on the prototype development, initial versions of a Base Agent, a Directory Agent, a Reconnaissance Agent, an Agent Guard, and the Deployment Center were baselined. Each component was continually improved through refinement and addition of new features.

At this point, we introduced a new hardware-based security implementation (Datakey) into the framework. The cost issue of the hardware security devices was raised. We investigated other possible implementations of PKCS-11 because the Luna solution was expensive, and found that Datakey was an acceptable solution.

We created a more formal testbed in our Lab, and we devised a series of phases to build the prototype framework. Each phase incorporated some additional feature, or added an option to

one or more SIDF components. This approach provided us with a flexible environment with which we could test each new build of a component.

Several messages are sent between components of the SIDF and the message types, along with the syntax and number of them, may need to be changed periodically. This could be problematic, since the message parsing modules would need to be rewritten when message changes are made. We created a message grammar that makes a table-driven parsing routine for messages. This feature provided for expansion and flexibility of framework component messaging. The message grammar is described in detail in the *SIDF Agent Guide for Development Specification* [SIDF-AGDS-TM-99-0003-Rev00-(9904)].

We continued to add features and functionality to each component of the framework. The testbed (which consisted of four computers) was used to test new functionality of the components.

One of the final steps was to integrate the Linux components into our testbed along with all the other SIDF components including versions of Base Agents, Directory Agents, Reconnaissance Agents, Analysis Agents, Decision-Response Agents, Agent Guards, and the Deployment Center.

We currently have a set of components that can be used for testing and operation of the SIDF. These components represent the baseline of our SIDF prototype development. Section 4.2 lists suggested changes and improvements that could be made to the SIDF.

3.2 SIDF Refinement

The future of the SIDF lies in the hands of the IDS developers. Through use, the future SIDF maintainers and developers will find and improve the current limitations of the SIDF, further adapting it for needs of intrusion detection and other environments. Several suggested refinements can be found in Section 4.2.

4.0 Results and Discussion

The following sections list this effort's final results. Since this effort was mainly a design and development effort, the bulk of the results consist of documentation, application source code, and executable files.

The documentation, source code, and executable files are delivered with this final report on CD-ROM.

4.1 Deliverables

The deliverables on this contract consist of documentation and applications. These items have been created and delivered during this contract:

- Framework Documentation
- Deployment Center
- NT Agent Guard
- Base Agent
- Directory Agent
- Decision-Response Agent
- NT Analysis Agent
- Linux Analysis Agent
- Ethernet Capture Reconnaissance Agent
- TCP/IP Display Reconnaissance Agent
- Traffic Analysis Reconnaissance Agent
- Immunology-base Reconnaissance Agent
- Linux Reconnaissance Agent
- Storage Agent

The following sections explain each of the bulleted items in the above list.

4.1.1 Framework Documentation

An extensive set of project documentation was written to describe, design, and develop the SIDF framework. The following table lists each of the documents delivered:

Table 1. SIDF Documentation Set

Document Title	Document Number
Scalable Intrusion Detection and Response Framework System Specification	SIDF-SS-TM-97-0032-Rev00-(9712)
Scalable Intrusion Detection and Response Framework Architecture Specification	SIDF-AS-TM-98-0002-Rev00-(9803)
Scalable Intrusion Detection and Response Agent Interface Control Document	SIDF-ICD-TM-98-0003-Rev00-(9803)
Scalable Intrusion Detection and Response Framework Detailed Design document	SIDF-FDD-TM-98-0004-Rev00-(9804)
TCPSniffer Design document	N/A
Unix Reconnaissance Agent/NT Proxy Agent Design document	N/A
An Open Infrastructure for Scalable Intrusion Detection	TM-98-0023
Scalable Intrusion Detection and Response Framework Software Test Description	SIDF-STD-TM-99-0002-Rev00-(9904)
Scalable Intrusion Detection and Response Framework Agent Guide for Development Specification	SIDF-AGDS-TM-99-0003-Rev00-(9904)
Scalable Intrusion Detection and Response Framework Final Report	SIDF-FRPT-TM-99-0004-Rev00-(9904)

A softcopy of each of these documents is provided on the CD-ROM delivered with this report.

4.1.2 Deployment Center

The Deployment Center initiates the framework, configures, issues and maintains the encryption keys for the other framework hosts, and controls the deployment of agents to other hosts. A copy of the main Deployment Center screen follows. Additional screen images and descriptions of all Deployment Center functions can be found in the *SIDF Software Test Description* [SIDF-STD-TM-99-0002-Rev00-(9904)].

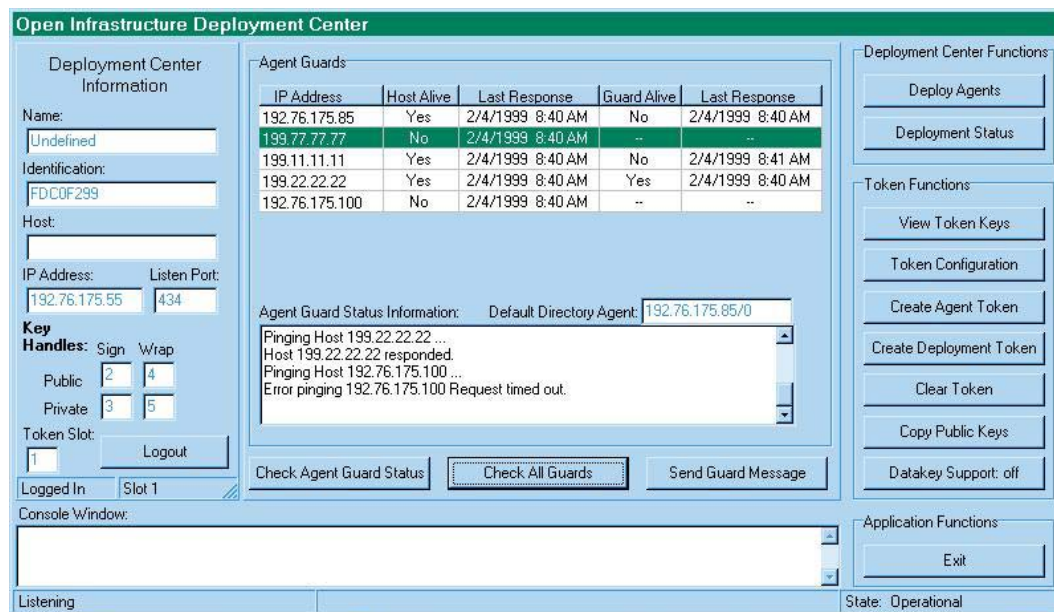


Figure 8. Deployment Center Main Screen.

4.1.3 Agent Guard

The Agent Guards receive deployed agent executables, initiates the execution of the agent on the host, and controls all local and external framework messaging. A copy of the main Agent Guard screen follows. Additional screen images and descriptions of all Agent Guard functions can be found in the *SIDF Software Test Description* [SIDF-STD-TM-99-0002-Rev00-(9904)].

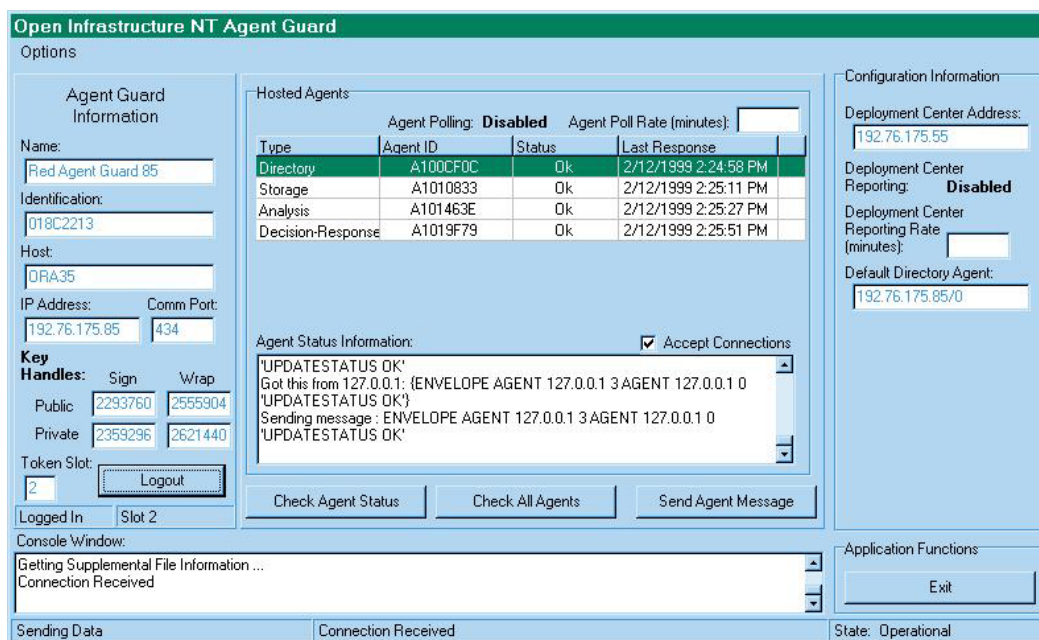


Figure 9. Agent Guard Main Screen.

4.1.4 Directory Agent

The Directory Agent maintains a list of each agent deployed in the framework. Additionally this agent stores the public keys of the other hosts in the framework. A copy of the Directory Agent screen follows.

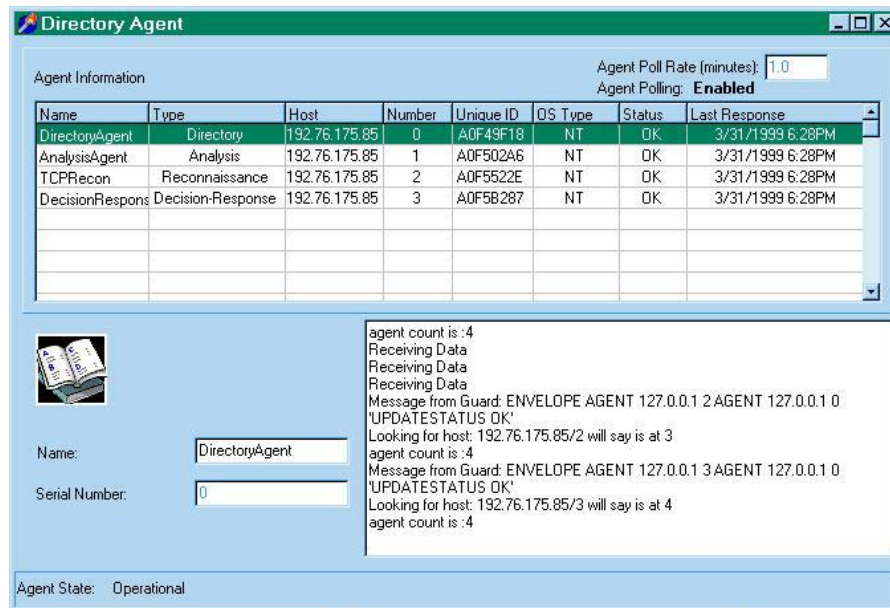


Figure 10. Directory Agent.

4.1.5 Decision-Response Agent

The Decision-Response Agent receives reports from one or more Analysis Agents and can be configured to perform user-initiated reactionary responses based on information received. A copy of the Decision-Response Agent screen follows.

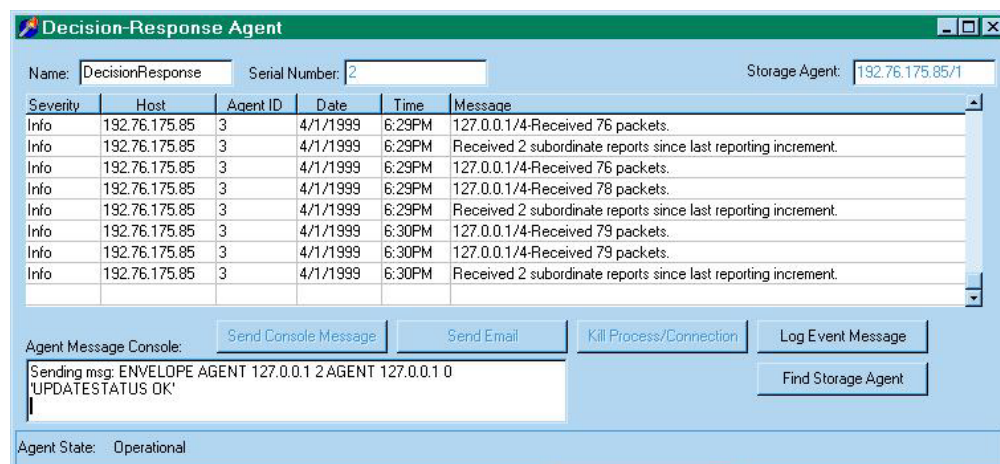


Figure 11. Decision-Response Agent.

4.1.6 Analysis Agents

The Analysis Agent receives reports from one or more Reconnaissance Agents. The Analysis Agents need to “understand” the information that a Reconnaissance Agent sends. Thus, the pairing of Analysis Agents to Reconnaissance Agents should be based on similar monitoring activities (e.g., TCP/IP Reconnaissance Agents should be paired with TCP/IP Analysis Agents). The Analysis Agents also report information periodically to a Decision-Response Agent. Two Analysis Agents have been built during this effort.

4.1.6.1 NT-based Analysis Agent

This first Analysis Agent runs on NT and specifically communicates with the three NT Reconnaissance Agents. It issues reports to the Decision-Response Agent. This agent’s interface is not very detailed (and is not presented here) because the main function of this agent is to collect data for multiple Reconnaissance Agents.

4.1.6.2 Linux-based Analysis Agent

This Analysis Agent is NT-based, but it communicates with the Linux Reconnaissance Agent. This agent configures, controls, and receives reports from the Linux Reconnaissance Agent. A copy of the Linux-based Analysis Agent screen follows.

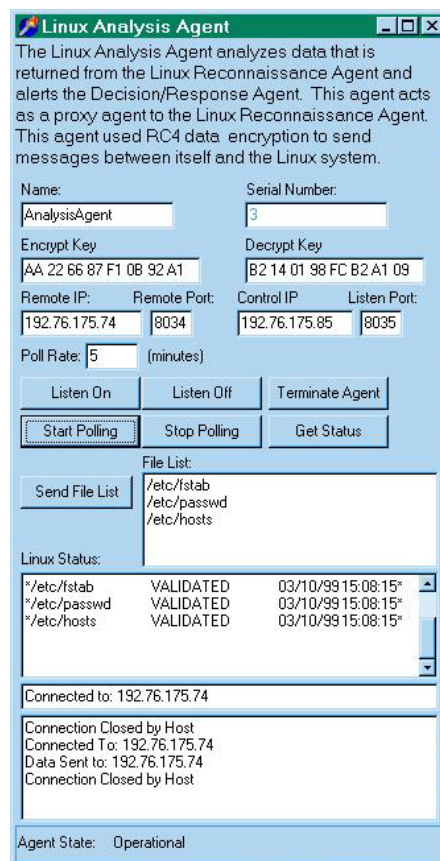


Figure 12. Linux Analysis Agent.

4.1.7 Reconnaissance Agents

Reconnaissance Agents collect information, perform a preliminary analysis, and report significant findings to Analysis Agents. A Reconnaissance Agent can be designed to be as specialized as a developer desires it to be. It can collect and report a variety of information to an Analysis Agent. The type, reporting interval, and quantity of data reported depends on how much data reduction is needed and what will be done with it. Reconnaissance Agents generally collect host state information such as the TCP/IP packets received, process information, event log data, system status, resource utilization, etc. We developed four Reconnaissance Agents on this project. These Reconnaissance Agents are presented as samples of the kind of functionality a Reconnaissance Agent could perform. Although these agents are presented here, they should not be viewed as complete ID components. Developers must review their needs and scope the functionality of each agent appropriately.

4.1.7.1 TCP/IP Display Reconnaissance Agent

This Reconnaissance Agent displays information associated with the Ethernet traffic passing through the network segment to which the host system is connected. This agent uses the TCPSniffer stack to capture TCP/IP packets. Although many Reconnaissance Agents will operate without operator guidance, this agent is an interactive agent. Buttons and fields are available to control the agent. The agent can also be controlled remotely with SIDF commands from an Agent Guard or an Analysis Agent. This agent captures each TCP/IP packet that passed on the network, and then it parses and displays each field of the packet. The user specifies the level of detail for display. For example, packet content can be displayed if desired--the agent is configurable. This means that IP addresses and ports can be entered to select the types of packets displayed. A copy of the agent's screen follows.

TCP Reconnaissance Agent

Reconnaissance Agents collect TCP/IP information and reports it to an Analysis Agent.

Agent Name: Agent Identification:

☐ Use Filters
 ☒ Automatically get received packets

☐ Lookup IP Address

Source IP/Port:
 Destination IP/Port:

Packet Date:
 Packet Time:

Ethernet Header - 14 bytes

TCP/IP Size:
 Source DNS:
 Destination DNS:

TCP Sequence:
 1FC75932:

Total Packets Sniffed:

Stopped

Receiving Data
 Receiving Data

☐ Save TCP Data

Data - 0 bytes

Adaptor Info
 Adapter Name: \Device\NPF{...} Adapter Medium: 802.3 Capture Address: 00:60:97:59:60:9E

Agent State: Operational

Figure 13. TCP/IP Display Reconnaissance Agent.

4.1.7.2 Traffic Analysis Reconnaissance Agent

The Traffic Analysis Reconnaissance Agent analyzes and reports to an Analysis Agent such network traffic data as the number, type, and content of packets that it has monitored during a specified time period. This agent uses the TCPSniffer stack to capture TCP/IP packets. A copy of the agent's screen follows.

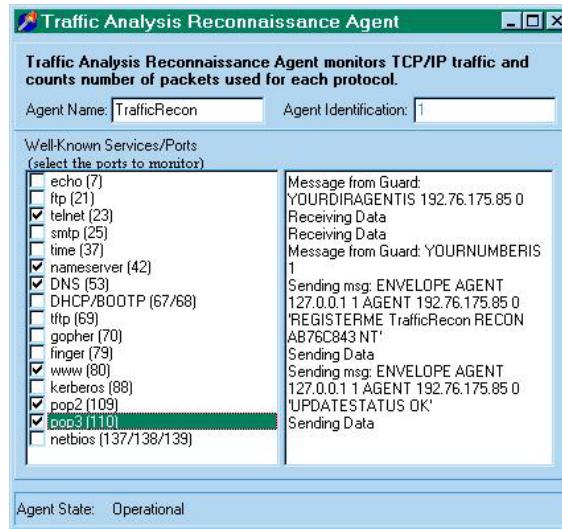


Figure 14. Traffic Analysis Reconnaissance Agent.

4.1.7.3 Immunology-based Reconnaissance Agent

The immunology-based Reconnaissance Agent monitors incoming packets and compares the packet content to a self-database. Packets that do not meet the criteria of the database will be flagged and a message sent to an Analysis Agent. This agent uses the TCPSniffer stack to capture TCP/IP packets. A copy of the agent's screen follows.

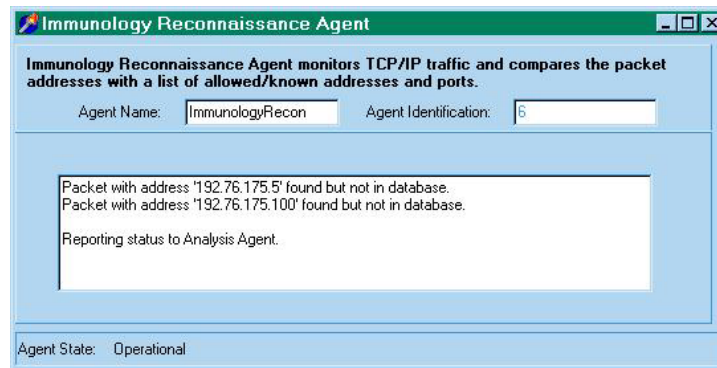


Figure 15. Immunology Reconnaissance Agent.

4.1.7.4 Linux Reconnaissance Agent

This Reconnaissance Agent runs under Red Hat Linux v5.2. This agent communicates directly to a Linux Analysis Agent running on an NT computer. This analysis agent can configure and control the Linux Reconnaissance Agent. The Linux Reconnaissance Agent monitors changes to system-specific files and reports status information to the Analysis Agent. This application is nongraphical.

4.1.8 Storage Agent

The Storage Agent acts as an event-logging agent for the framework. Any SIDF component can issue a message to be logged by a Storage Agent. This application is nongraphical.

4.2 The Future of SIDF

As is the case with any software development effort, the technology and software applications cannot just be stored on a shelf. Development must continue, and the technology matured and refined. The SIDF represents a concrete technology adaptation for the use of agent technology within the IDS environment.

We have successfully implemented a prototype version of the SIDF with a set of example agents. Throughout the development, we found instances where improvements could be made to either the framework or the framework's components. Each suggested improvement will be discussed in the following sections. The benefits and associated implications are annotated with each suggestion.

4.2.1 Provide the Agent as an ActiveX Control

Instead of having developers modify Delphi control, the Base Agent, Directory Agent, etc. could be turned into ActiveX controls. Creation of ActiveX controls would enable us to use the agent's skeleton code in any environment that supports ActiveX integration. More importantly, this change would uncouple the development of new agents from the Delphi environment currently required. Other development environments could be used such as Java or Visual Basic.

It is envisioned that a developer would load the application development environment of his choosing and select the desired agent type from a palette of ActiveX agent controls. If he wanted to create an Analysis Agent, an "Analysis Agent" control would be placed on the developer's form. Access to the agent's properties would be conducted through the properties screen of the development environment. Customized event handlers could be implemented by exposing the message handling functions through the event properties screen.

This improvement would provide an independent development method for new agent implementation, and it should be performed once other improvements are made. The creation of an ActiveX control is not difficult. But installation of the control into a development environment is not as dynamic as just changing some code and recompiling the application. The ActiveX control usually needs to be unregistered, moved to a system directory, and installed. Therefore, we suggest that once the prototype framework has undergone some "field-testing," then creating ActiveX controls would be advisable.

4.2.2 Fault Tolerance Communication

Improvement in the fault tolerance of agent communication needs to be addressed. The framework uses TCP/IP for communication. The TCP/IP protocol guarantees delivery of the message, as opposed to other protocols (such as UDP), which do not. In addition, the framework uses buffered sockets, which ensures that the complete message is delivered. However, the current communication scheme does not guarantee that messages from one agent actually get to

another agent. Since one or two Agent Guards act as intermediaries in the communication between agents, the Agent Guard may not be able to deliver the message due to problems it encounters. In some cases, a message may be sent to the initiating agent. For example, a message sent from Agent 1 to Agent 2 on the same host will go through the local Agent Guard. Agent 1 sends the message to the Agent Guard. Next the Agent Guard tries to deliver the message to Agent 2. If Agent 2 is not responding or has terminated, the Agent Guard will send a message back to Agent 1 indicating the termination of Agent 2. But if the Agent Guard does not receive any errors from Agent 2 and Agent 1 does not receive any errors or notification from the Agent Guard, then the message is assumed to have been delivered to Agent 2.

A simple addition to the messaging protocol would be a requirement that an acknowledgement be sent for every message sent in the framework. A couple of problems surface with this suggestion's implementation, namely the message traffic in the framework will double, and there must be a way to associate acknowledgements with messages.

4.2.3 Improve Security of Communication

The security of framework communication needs to be upgraded. Although framework communication is encrypted, it is still susceptible to attack. Since the Agent Guard and the Deployment Center are listening on TCP ports, they are susceptible to all the same kind of network-based port flooding attacks as any other network-based system. All the communications that the Agent Guards and the Deployment Center obtain from the TCP ports are encrypted. However, to determine if the message is valid, it first is collected from the socket, and an attempt is made to decrypt it. Noticeably invalid messages will be discarded, but useless processing of the message will have taken place.

Possible solutions to a flooding attack or denial of service would be to only accept connections from known addresses. At least the Deployment Center could be protected in this fashion since it has a list of all known hosts that will participate in the framework. Agent Guards could be designed to modify the list of allowed hosts when new encryption keys are obtained. Of course, this does not preclude the possibility of an attack of the framework from a host that is listed in the "allowed host" list.

A second possible solution would be to send a special message prelude or header that would be first examined when a new packet is starting to be received. If the message header is not as expected, then the connection would be quickly closed. After a predetermined number of invalid messages from a host, that host could be added to the denied host list.

4.2.4 Add More Specialized Reporting

Currently, only general reporting messages are defined that consist of a severity indicator and a message parameter. This messaging structure could be expanded to include very specific message types that a message number could identify. The message text could then consist of an arbitrary binary array that could be parsed based on the message code. This would introduce many more specialized messages without changing the message format.

In addition, the messages could be made to conform to the standards outside this project's current scope. Namely, we would suggest adding the message identifiers used for the CIDE GIDOs.

4.2.5 Optimize the Code and the Libraries

Much of the prototype code is redundant between agents, and we should create libraries to be linked into the applications. This way, only one copy of the common code will need to be maintained.

4.2.6 Software-based PKCS-11 Implementation

The SIFE framework is heavily based on the PKCS-11 standard and hardware encryption. We have decreased the cost for each host in the SIFE from \$1000 per host down to less than \$100 per host by supporting Datakey devices in the framework.

A software-based PKCS-11 implementation could decrease the cost of deploying the framework by \$1,000 from the framework control side (the Deployment Center) and by \$100 per each addition SIFE host. We would suggest using a floppy/removable disk as the "token" within a software-based PKCS-11 implementation since the framework control would only have to be modified slightly.

This option would allow the interspersing of differing types of security devices within the framework--PCMCIA tokens, Datakeys, and software keys.

Some problems exist that preclude its immediate implementation. First, a commercial company must release a software-based PKCS-11 implementation. We have used Wetstone Technologies' SMARTCrypt implementation and would like to continue with it. Second, the software implementation must support the functions that we have currently implemented with the Luna and Datakey tokens. Specifically, RSA algorithms must be included.

4.2.7 Add Agent Types

Additional types of agents should be added to the framework. For example, several types of Analysis Agent skeletons could be created and stored in the software repository. Each would have a specialized set of functions to receive data from similar typed Reconnaissance Agents. An Analysis Agent could be built to process NT event log data or another could specifically handle TCP/IP data. This would allow each future developer access to a set of specialized functions and would free them from reinventing a lot of this basic functionality with each new agent created.

4.2.8 Update Ethernet Capture Routines

The Ethernet capture components should be updated so that they don't depend on having an Ethernet packet-capture driver installed on the host that the TCP/IP Reconnaissance Agent is deployed. We used a capture library that is specific to NT. Much more dynamic libraries are now available. Changing this feature would require that the TCP/IP ActiveX control be rewritten or replaced with a different control that doesn't depend on the added protocol stacks.

4.2.9 Upgrade to Delphi 4

During the development on this contract, Delphi 4 was released (late 1998). Since we had substantial investment in the current version of Delphi and were not sure how easily the framework would port, we did not upgrade at that time. Additionally, several of the third-party components that we used may not be compatible with Delphi 4.

Delphi 4 has introduced several new features that could be useful within the framework and would reduce the number of third-party components used. The risk associated with this move to Delphi 4, range from the very simple task of recompiling each application to the extreme of having to rewrite major sections of the framework code. The available of alternative implementations of functions and controls in Delphi 4 must be investigated before committing to this transition.

4.2.10 Use fewer third party components

With the upgrade to Delphi 4, some third-party components may not be required. The availability of new ActiveX controls, the amount of effort to convert to new controls, and the additional functionality they might provide, needs to be investigated.

4.2.11 Create a Custom Palette

It is envisioned that a simpler way to develop new agents would be to have something similar to a “developer’s palette” for building agents. We would like to extend the SIDF so that it could be included into a variety of environments in addition to Delphi. From the user-selected environment, a palette of agents would be available. For example, within a tool such as Visual Basic, a developer could select “Reconnaissance Agent” and drop a Reconnaissance Agent widget onto his form. This action would create a Reconnaissance Agent, providing the developer access to all the properties, methods, and events of a SIDF Reconnaissance Agent. This concept would apply to all SIDF types of agents and could be expanded to include specialized agent type such as “Linux Audit Log Progressing Reconnaissance Agent” and “NT Privileged Process Monitor Reconnaissance Agent.”

4.2.12 Compress Files and Messages

No message or file compression is used within the SIDF. The added benefit of smaller messages and files would greatly decrease the amount of data transmitted between components and enhance overall framework throughput. The added processing time needed to perform data compression should be outweighed by the smaller amount of data that needs to be encrypted and transmitted.

4.2.13 Automatic Deployment

The current version of the SIDF needs to be manually configured. This is especially tedious when starting up the framework for the first time. A simple batch file format consisting of the name of the agent and the host to which to deploy the agent could be created. The Deployment Center could read a startup deployment batch file and process each entry when it starts. Another

less-automated approach is to allow the user to select a batch file for processing, so that each agent needn't be deployed manually one at a time.

4.2.14 Automatic SIDF State Archival

In certain instances, it is useful to know the current state of the SIDF (a listing of all deployed components and the status of each). Additionally, this state information could possibly be used to restore the SIDF to a previous state by comparing the current SIDF state with the previously saved one. The appropriate deployment, retraction, and configuration commands could then be automatically issued to set the SIDF state.

4.2.15 Automatically Find Associated Agent

Analysis Agents and Reconnaissance Agents need superior agents to which to report information. Decision-Response Agents and Analysis Agents need subordinate agents from which to receive and process information. If these agents did not have superior and/or subordinate agents, their functionality would be greatly impaired. Each agent, by the nature of its function, supplies or receives data.

In the currently framework, the definition of an agent's superior or subordinate is not encoded in the deployed agent at startup time. The Directory Agent does contain the list of available agents and a newly deployed agent could query the Directory Agent for a list of possible superior or subordinate agents. We don't do this, though, because the granularity of the query results are not sufficiently detailed to automatically select a "correct" superior/subordinate. For example, a TCP/IP Packet Reporting Reconnaissance Agent would not provide much useful data to an Analysis Agent that processes NT event logs.

4.2.16 Upgrade Directory Agent

The Directory Agent plays a very important role in the tracking the state of agents in the SIDF. The Directory Agent currently provides limited functionality in the ability to query for specific types of deployed-agent information. A substantial set of additional information could be stored for each registered agent. The Directory Agent grid defines some very basic information on each agent. The data maintained on each agent is initially passed to the Directory Agent at agent registration time. Additional agent-configuration data can be added to the registration message and new querying functions can be implemented for access.

The Directory Agent collects SIDF state information by periodically polling agents. The status messages returned by the agents simply indicate that the agent is still functioning but nothing more. Also, nonreporting agents are listed but the Directory Agent makes no attempt to investigate the reason why agents are not reporting, nor does the Directory Agent report this anomaly to a superior agent or the user.

4.2.17 Replay Attack Protection

SIDF communication could be susceptible to a replay attack. If a packet is captured, it could be resent to the receiving host computer(s). The contents of the packet could not be modified, just

repeated. The obvious answer to this type of attack is to place an incrementing counter into the packet. This alteration of the packet would prevent the possibility of this type of attack.

4.2.18 CIDE Incorporation

The DARPA Common Intrusion Detection Framework (CIDE) is currently being revised. We did not target the incorporation of CIDE into the SIDE initially for these reasons:

- CIDE is a proposed standard, whereas the SIDE was a proposed implementation.
- The SIDE was proposed approximately a year before the CIDE standard.
- The CIDE standard has evolved during the SIDE contract but still has not been finalized.
- We have provided input into the CIDE standard and several features are similar to those in the SIDE.
- CIDE has been proposed to the IETF as a standard. It may undergo substantial revision during this process.
- We initially were going to use the CIDE message structure in the SIDE, but felt that it was not mature enough at the time of our implementation.
- We have used a message format defined by a message grammar, which would allow us to change the message formats by simply modifying the message grammar.

We feel that CIDE incorporation is possible into the SIDE, but would suggest waiting until the CIDE has been standardized.

4.2.19 Send Initial Configuration Information

When an agent is designed, developed, and built, it is configured with a set of initialization parameters. If the application is run once or a dozen times, the startup configuration is identical each time (assuming that the application is not reading configuration parameters from different files). Since configuration files cannot be deployed with agent executables in the current SIDE implementation, the startup of a deployed agent will be the same each time. Messages can be sent to change the configuration of the agent once it is running. An improvement to the framework would be to allow the passing of a set of command-line parameters or some other method by which an agent could be configured upon initialization after it was deployed.

4.2.20 Deploy Additional Files with Agent

The Agent Executables Database consists of a directory of executable files. Each file was generated from an SIDE agent template, and it represents a standalone application that will be transmitted to another host in the framework. Sometimes, based on the application functionality or a developer's implementation, an application requires one or more external files. Data and configuration files are two likely types of files that might be required by an application. The current implementation of the SIDE does not provide a method for associating or transmitting supporting files with a deployed agent.

4.2.21 Add Auditing Functions

Messages can be logged to the Storage Agent at the discretion of the developer (for internally logged messages) or the user (for interactively specified messages). The SIDF does not automatically log events as they are generated. For a higher degree of auditing, functions should automatically log each operation that occurs in the SIDF. Levels of auditing can be specified to differentiate user-initiated vs. system-generated actions.

4.2.22 Framework State Tracking

The Directory Agent can track the SIDF's state at a particular instance. What the Directory Agent lacks are methods to track changes to the framework's state. A log or database could be set up that would indicate the cumulative status of the framework. Information indicating the sequence of agent registering, reporting, and unregistering would be saved.

4.2.23 Framework Traceability

From the perspective of a forensic investigation tool, the SIDF does not have a very structured logging capability. The addition of detailed logging and querying capabilities would provide information needed in a forensic investigation. A framework snapshot would need to be maintained at periodic increments and as requested. A log would show the actions and results that occurred in the framework. A snapshot would show the current status of the framework (list of agents deployed) at the snapshot time. Additional querying would return specific information on agent state and health. It would also be important for the system to document all details, especially negative details. For example, the knowledge that a particular agent was *not running* at a particular time could be more important than the list of agents that were running.

4.2.24 Add Database Capabilities

Database support could easily be added to the SIDF, and it would enhance data retrieval functions designed into SIDF agents and framework-control applications. Currently only direct text-file access is supported. Database functions would be especially useful for querying stored data that was collected by Reconnaissance Agents or processed by Analysis Agents.

4.2.25 Agent Information Archival

Instead of disposing of previously collected information (information saved from a previous deployment of the same agent), it would be possible for an agent to continue collecting information and posting it with the previous information collected. This feature would be of use when a specific agent is deployed to a host, then retracted and later redeployed to the same host.

4.2.26 Reduce Agent Guard Visual Component

The current Agent Guard has been designed with a very robust user interface. Mainly the interface has been used to test the functionality of agents and to provide visual indication of SIDF message transmission. The Agent Guard could be revised to reduce and possibly remove the user interface. It is envisioned that the Agent Guard would be created as an NT service that would be automatically started when a machine is rebooted. The encryption functions would

still require the entry of a password that should not be stored locally on the host. Within the implementation of this enhancement, a method of securely logging in the Agent Guard token would have to be addressed.

4.2.27 Agent Visual Component Modification

As with the Agent Guard, most agents are dependent on a visual user interface. Some agents use the interface more extensively than others. The basic format of the agent's main screen is inherited from the Base Agent. Although the Base Agent's default screen does not add much to the operation of the derived agents, a minimal interface is required to place the nongraphical runtime components. Possibly the graphical interface of the Base Agent could be removed, or a selection of main interface screens created so that a developer could select the most appropriate screen.

4.2.28 Reduce Agent Dependency

Each agent is dependent on its source files, shared source files, basic type of agent files (i.e., a TCP/IP Reconnaissance Agent depends on files used to build a generic Reconnaissance Agent), Base Agent project files, and SIDS message files. Any change to a single one of these files requires each agent to be recompiled. Based on the current method of inheritance, this problem cannot be easily corrected, but it can be reduced.

Since the SIDS is in a state of being modified to enhance its capabilities, several source files are routinely changed which in turn requires SIDS components to be rebuilt. It is possible that some commonly used functions could be moved to a dynamically linked library (DLL). This action would allow some modification of SIDS without requiring component rebuilding.

4.2.29 Add Deployment Management Console

The Deployment Center allows deployment of individual agents. A previously suggested enhancement would allow batch automatic deployment of agents. Yet, in the configuration of a system it is difficult for a system administrator using the Deployment Center to get a clear understanding of the SIDS configuration and know what is deployed on which hosts at a particular time.

We suggest adding a Deployment Management Console to the Deployment Center. The Deployment Management Console would be a screen that presents a graphical view of the SIDS (a network map) with icons representing the hosts with interconnecting lines representing the network, and boxes representing the agents. From the SIDS map, host and agent status would be retrieved by clicking on an icon. Another feature would be the ability to deploy a new agent dragging an agent icon onto the desired host. This modified view of the Deployment Center functionality would improve the overall operation of SIDS deployment and framework status.

5.0 Conclusions

As was listed in the previous sections, the SIDF project has successfully proven itself. The next step is to continue the use of this technology in new programs.

5.1 Program Technologies

The following is a list of the technologies and features incorporated into the SIDF:

- Multiple Applications Developed
- Multiple Operating Systems in Framework
- Multiple Hosts (2-3 Windows NT 4.0 and one Red Hat Linux v5.2)
- Multiple Processes Running on each Host
- Framework Interprocess Communication
- Security Standard--PKCS-11
- Two types of Encryption Tokens (Luna/Datakey)
- Delivery of Software Development Documents [over 500 pages]
- Delivery of over 12 different pieces of software, including Reconnaissance Agents, Deployment Center, Agent Guard, etc.
- Delivery and Integration of both Unix and NT components
- Delivery of low level socket communication and TCP/IP components

5.2 Future Programs

Odyssey Research is actively pursuing additional opportunities that will make use of the SIDF and the technologies developed. We continue to team with Wetstone Technologies on new programs that make use of hardware-encryption devices. And we have continued to invest in the SIDF program by including it as part of other on-going efforts including the Omnisleuth forensics project as well as writing new proposals that incorporate SIDF as the baseline technology used.

6.0 References

The following documents serve as references throughout this project:

- [Bis92] Matt Bishop. A Model of Security Monitoring. Department of Math and Computer Science, Dartmouth College, 1992.
- [Bis93] Matt Bishop. A Standard Audit Trail Format. Department of Math and Computer Science, Dartmouth College, 1993.
- [Che94] William R. Cheswick and Steven M. Bellovin. *Firewalls and Internet Security*. Addison-Wesley, 1994.
- [Den96] Dorothy E. Denning. Protection and Defense of Intrusion. Georgetown University, March 1996.
- [Far94] Daniel Farmer and Eugene H. Spafford. The COPS Security Checker System. Purdue University Technical Report, CSD-TR-993, January 1994.
- [For94] Stephanie Forrest and Dipankar Dasgupta. Novelty Detection in Time Series Data using Ideas from Immunology. Department of Computer Science, University of New Mexico, 1994.
- [For95] Stephanie Forrest, Alan Perelson, Lawrence Allen, and Rajesh Cherukuri. A Change-Detection Algorithm Inspired by the Immune System. *IEEE Transactions on Software Engineering*, February 1995.
- [For96] Stephanie Forrest, Patrik D’haeseleer, and Paul Helman. An Immunological Approach to Change Detection: Algorithms, Analysis and Implications. Department of Computer Science, University of New Mexico, 1996.
- [Hlm90] L.T. Heberlein, K. N. Levitt, and B. Mukherjee. A Method to Detect Intrusive Activity in a Networked Environment. *Proceedings of the 14th National Computer Security Conference*, pp. 362–371, October 1991.
- [Hoa95] James Hoagland, Christopher Wee, and Karl Levitt. Audit Log Analysis Using the Visual Audit Browser Toolkit. UC Davis Computer Science Technical Report, September 1995.
- [Hos96F] Chester D. Hosmer, Jr. Computer and Computer System Auditing and Intrusion Detection Monitoring, Future Development Forecast Analysis. NY State Technology Enterprise Corporation & Odyssey Research Associates, September 1996.
- [Hos96S] Chester D. Hosmer, Jr. Computer and Computer System Auditing and Intrusion Detection Monitoring, State-of-the-Art Technology Survey. NY State Technology Enterprise Corporation & Odyssey Research Associates, September 1996.
- [Kim93] Gene H. Kim and Eugene H. Spafford. The Design and Implementation of Tripwire: A File System Integrity Checker. Purdue University Technical Report CSD-TR-93-071, November 1993.

- [Kim94] Gene H. Kim and Eugene H. Spafford. Writing, Supporting, and Evaluating Tripwire: A Publicly Available Security Tool. COAST Laboratory, Department of Computer Science, Purdue University, March 1994.
- [Kum95] Sandeep Kumar. Classification and Detection of Computer Intrusions. Doctoral Thesis, August 1995.
- [Lev95] Karl N. Levitt, Steven Cheung, and Calvin Ko. Intrusion Detection for Network Infrastructures. Department of Computer Science, University of California Davis, 1995.
- [Lev96] Steven Levy. Wisecrackers. *Wired Magazine*, pp. 128+, March 1996.
- [Max90] Roy A. Maxion and Frank E. Feather. A Case Study of Ethernet Anomalies in a Distributed Computing Environment. Carnegie Mellon University, October 1990.
- [Max93] Roy A. Maxion and Robert T. Olszewski. Detection and Discrimination of Injected Network Faults. School of Computer Science, Carnegie Mellon University, June 1993.
- [Neu94] Michael Neuman. Monitoring and Controlling Suspicious Activity in Real-time with IP-Watcher. En Garde Systems, 1994.
- [Riv92] R.L. Rivest. RFC 1321: "The md5 message-digest algorithm." Technical report, Internet Activities Board, April 1992.
- [Soh95] B.C. Soh and T.S. Dillon. Setting optimal intrusion-detection thresholds. *Computers & Security*, Vol. 14, No. 7, 1995.
- [Spa91] Eugene H. Spafford. OPUS: Preventing Weak Password Choices. Purdue University Technical Report CSD-TR-92-028, June 1991.
- [Spa94] Gene Spafford and Mark Crosbie. Defending a Computer System using Autonomous Agents. Department of Computer Science, Purdue University, February 1994.
- [Spa95] Gene Spafford and Mark Crosbie. Active Defense of a Computer System using Autonomous Agents. Department of Computer Science, Purdue University, February 1995.
- [Ste94] Richard W. Stevens. *TCP/IP Illustrated, Volumes 1, 2 & 3*. Addison-Wesley Publishing Company, March 1996
- [Sto96] Salvatore J. Stolfo and Philip K. Chan. On the Accuracy of Meta-learning for Scalable Data Mining. Department of Computer Science, Columbia University, 1996.
- [Win90] J.R. Winkler and W.J. Page. Intrusion and Anomaly Detection in Trusted Systems. Planning Research Corporation, Government Information Systems, 1990.